Programmeerimise põhikursus

ITI0010

Loengu sisu

- Progr in the large: funktsioonid:
  - Funktsioonide mõte
  - Modifitseerijad, tüübid, argumendid jne
- Progr in the large: etteruttavalt klassidest
- Andmed:
  - Kordamine: lihttüübid (int, float, etc)
  - “Keerukad tüübid” objektidena: mis nad sisuliselt on
  - Stringid
  - Muud struktuurid objektidena
  - Massiivid

Prog. in the large: grouping and re-use of code

- Sometimes you need a similar piece of code in several parts of your program.
- If you have a lot of code, it is important to group code into clear funtional units.
- You may want to use these functional units in several places, not just once.
- In Java, there are two ways to group and reuse parts of code:
  - Functions
  - Classes

A re-usable chunk of code

A set of functions grouped together, normally in one file

Subroutines (functions)

- The main method for breaking up a complex program into manageable pieces is to use subroutines.

- A subroutine consists of the instructions for carrying out a certain task, grouped together and given a name. Elsewhere in the program, that name can be used as a stand-in for the whole set of instructions.

- As a computer executes a program, whenever it encounters a subroutine name, it executes all the instructions necessary to carry out the task associated with that subroutine.

Functions

- Some subroutines are designed to compute and return a value. Such subroutines are called **functions**.
- You've already seen examples of mathematical functions such as Math.sqrt(x) and Math.random()
- If you write a method that is meant to be used as a function, then you have to specify the return-type of the method to be the type of value that is computed by your function. For example, the return-type of boolean in

  public static **boolean** lessThan(double x, double y)

  specifies that lessThan is a function that computes a boolean value. The return type of Math.sqrt() is double.

- A method that is meant to be used as an ordinary subroutine, rather than one that returns a value, must specify void as its return-type. The term "**void**" is meant to indicate that the return value is empty or non-existent.

Functions instead of copy...paste

- Sometimes you need a similar piece of code in several parts of your program.

```
x=1.2;
y=(sin(x*2)-cos(4*x))+1.5*cos(x-0.5);
z=2.3;
u=(sin(x*2)-cos(4*x))+1.5*cos(x-0.5);
```

- Use functions instead of copy-paste:

```
double myfun (double z) {
  return (sin(z*2)-cos(4*z))+1.5*cos(z-0.5);
}
x=1.2;
y=myfun(x);
z=2.3;
u=myfun(z);
```

Functions instead of copy...paste

- Instead of

```
for(i=0;i<m.length; i++) m[i]=0;
for(j=0;j<r.length;j++) r[j]=0;
```

- Use

```
void clear(int[] x) {
  int i;
  for (i=0;i<x.length;j++) {
    r[i]=0;
}
....
clear(m);
clear(r);
```

Function

- A function definition in Java takes the form:

```
modifiers  return-type  function-name  ( parameter-list ) {
        statements
}
```

- There can be several modifiers
- It is not obligatory to have any modifiers

Modifiers are important for programming with classes and objects. They are not Important if we do not use object-oriented style

- Parameter-list contains variables together with their types

- NB! Functions are sometimes called methods. This has to do with classes and objects as we will see later.

Examples

```
static void playGame() {
        // "static" is a modifier; "void" is the return-type;
        // "playGame" is the method-name; the parameter-list is empty
        . . .  // statements that define what method here
}
 int getNextN(int N) {
     // there are no modifiers; "int" in the return-type
     // "getNextN" is the method-name; the parameter-list  includes
     // one parameter whose name is "N" and whose type is "int"
     . . .  // statements that define what method does go here
}
public static boolean lessThan(double x, double y) {
    // "public" and "static" are modifiers; "boolean" is the return-type;
    // "lessThan" is the method-name; the parameter-list  includes
    // two parameters whose names are "x" and "y", and the type
     // of each of these parameters is "double"
     . . .  // statements that define what method does go here
}
```

Use static unless you are actually using object-oriented style of programming.

Static functions behave like normal C functions.

They do not always mix well with non-static functions!

Function return values

- Computed value is given back by a return statement, which takes the form:

```
return expression;
```

- Example:

```
static double pythagorus(double x, double y) {
  // Computes the length of the hypotenuse of a right
  // triangle, where the sides of the triangle are x and y.
  return  Math.sqrt(x*x + y*y);
}
```

Function return values

- A very simple function that could be used in a program to compute 3N+1 sequences. Given one term in a 3N+1 sequence, this function computes the next term of the sequence:

```
static int nextN(int currentN) {
   if (currentN % 2 == 1)      // test if current N is odd
      return 3*currentN + 1;   // if so, return this value
   else
      return currentN / 2;     // if not, return this instead
}
```

- Exactly one of the two return statements is executed to give the value of the function.
- A return statement can occur anywhere in a function.

Classes as groups of functions

- When you define a subroutine, all you are doing is telling the computer that the subroutine exists and what it does. The subroutine doesn't actually get executed until it is called.
- For example, the playGame() method defined above could be called using the following subroutine call statement:

playGame();

- To call playGame() from outside the Poker class, you would have to say

Poker.playGame();

- More generally, a subroutine call statement takes the form

function-name(parameters);

if the function that is being called is in the same class, or

class-name.function-name(parameters);

else

Large program components: Classes

- When i call a function

```
System.out.println("something")
```

Then Java will, roughly said (but this is an oversimplification)

  - Try to locate class called “System”
  - Look for the class “out” inside class “System” (partly true only!)
  - Look for the function “println” inside the class “out”

- Same happens when we call, say, TextIO.readln():

  - “TextIO” class will be located,
  - “readln” function will be searched for from this class

Function parameters

- If a subroutine is a black box, then a parameter provides a method for passing information from the outside world into the box.
- Parameters allow you to customize the behavior of a subroutine to adapt it to a particular situation.

```
static void Print3NSequence(int startingValue) {
    int N = startingValue;
    int count = 1;
    TextIO.putln("The 3N+1 sequence starting  from " + N);
    TextIO.putln();
    TextIO.putln(N);
    while (N > 1) {
        if (N % 2 == 1)      // is N odd?
          N = 3 * N + 1;
        else
          N = N / 2;
        count++;   // count this term
        TextIO.putln(N);  // print this term
     }
     TextIO.putln();
     TextIO.putln("There were " + count + " terms in the sequence.");
}  // end of Print3NSequence()
```

Function parameters

- This subroutine could be called using statements such as

```
Print3NSequence(17);
```

  which would print out the 3N+1 sequence starting from 17.

- If `K` is a variable of type `int`, then the statement

```
Print3NSequence(K);
```

  would print out the 3N+1 sequence starting from `K`, that is from what ever happens to be the value of `K` when the statement is executed.

Function parameters

- Consider, for example, a subroutine

```
static void doTask(int N, double x, boolean test) {
     // statements to perform the task go here
 }
```

- This subroutine might be called with the statement

```
doTask(17, Math.sqrt(z+1), z >= 10);
```

Function parameters

- When the computer executes this statement, it has essentially the same effect as the block of statements:

```
{
  int N = 17; // declare an int named N
              // with initial value 17
  double x = Math.sqrt(z+1);
            // compute Math.sqrt(z+1),
            // use it to initialize a
            // new variable x of type
            // double
  boolean test = (z >= 10); // evaluate
             // "z >= 10"
            // and use the resulting
            // true/false value to
            // init a new variable
 // statements to perform the task go here
}
```

Function parameters

- There are actually two different sorts of parameters:
  - Formal parameters or dummy parameters the parameters that are used in the definition of a subroutine
  - Actual parameters or arguments: the parameters that are passed to the subroutine when it is called.
- A formal parameter must be an identifier, that is, a name. A formal parameter is very much like a variable, and -- like a variable -- it has a specified type such as `int`, `boolean`, or `String`.
- An actual parameter is a value, and so it can be specified by any expression, provided that the expression computes a value of the correct type.

Function parameters

- Java (and C++) allow two different subroutines in the same class to have the same name, provided that their signatures (argument lists) are different.
- We say that the name of the subroutine is overloaded because it has several different meanings.
- The computer doesn't get the subroutines mixed up. It can tell which one you want to call by the number and types of the actual parameters that you provide in the subroutine call statement.
- Example: TextIO:

```
putln(int)          putln(int,int)          putln(double)
putln(String)       putln(String,int)       putln(char)
putln(boolean)      putln(boolean,int)      putln()
```

- NB! The signature does not include the subroutine's return type. It is illegal to have two subroutines in the same class that have the same signature but that have different return types.

Programming in the large II: objects, classes

- The reusable components should be as "modular" as possible. A module is a component of a larger system that interacts with the rest of the system in a simple, well-defined, straightforward manner.
- The idea is that a module can be "plugged into" a system. The details of what goes on inside the module are not important to the system as a whole, as long as the module fulfills its assigned role correctly.
- This is called **information hiding**, and it is one of the most important principles of software engineering.
- A module (a typical class) is a “black box” with mostly everything hidden from other programmers.

Large program components: Classes

- A program is made up of classes. To write a program, you have to write at least one class, and that class must include a main() routine. Applets are similar, except that the class you write doesn't need a main() routine.

- However, every non-trivial program or applet makes use of other classes as well. In this chapter, you have seen programs that use classes called TextIO and System.

- Whenever a program or applet makes use of a class, the Java interpreter will go off and try to locate that class. The first place it looks is in the same directory from which the main program or applet class was loaded. If it doesn't find it there, it will search elsewhere. Exactly where it looks depends on the particular version of Java that is running.

- You can depend on certain standard classes, such as System, being available in any version of Java.

Large program components: Classes

- When i call a function

```
System.out.println("something")
```

Then Java will, roughly said (but this is an oversimplification)

  - Try to locate class called “System”
  - Look for the class “out” inside class “System”
  - Look for the function “println” inside the class “out”

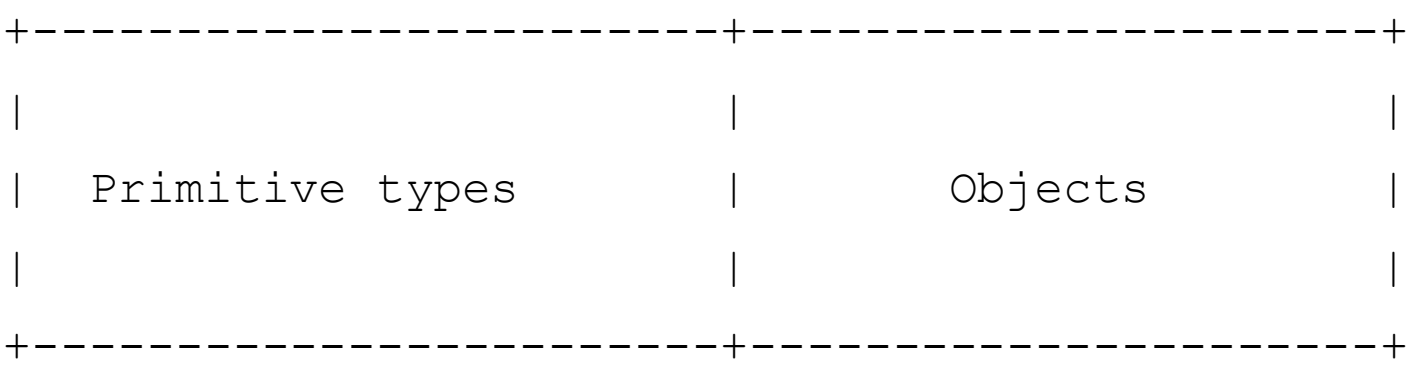
- Same happens when we call, say, TextIO.readln():

  - “TextIO” class will be located and “readln” function will be searched for from this class

Data

- Java has *very many* data types built into it, and you (as a programmer) can create as many more as you want.

- However, other than the primitive data types, *all the other data in a Java program will be represented as an object*. So there is a fundamental split in the data a Java program deals with:



All Data

Primitive TYPES

- The primitive types are named:

```
byte, short, int, long,
float, double,
char,
boolean
```

- short corresponds to two bytes (16 bits). Variables of type short have values in the range -32768 to 32767.
- int corresponds to four bytes (32 bits). Variables of type int have values in the range -2147483648 to 2147483647.
- long corresponds to eight bytes (64 bits). Variables of type long have values in the range

  -9223372036854775808 to 9223372036854775807.
- **`boolean`** `is result of: rate > 0.05`

What is a complex type?

- Basically, a **complex type** is a place to store several simple types, one after another. You can then use this “data object” as one whole.

- Examples:

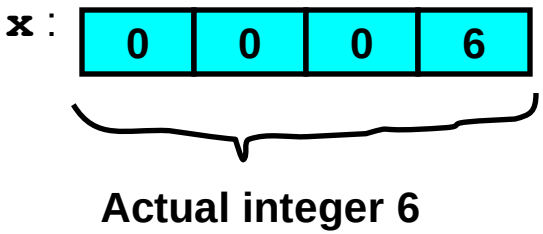
  - **String**: several simple characters one after another: “asasad adsas”
  - **Array**: like a string, but may contain integers, floats or any other type of data (also complex types)!
  - **A structure** representing a person: contains, for example
    - A string holding the name: “Jaan Mets”
    - An integer holding age: 21
    - A float holding length: 1.83

Where are complex objects in memory?

- If you use a simple type, like integer:

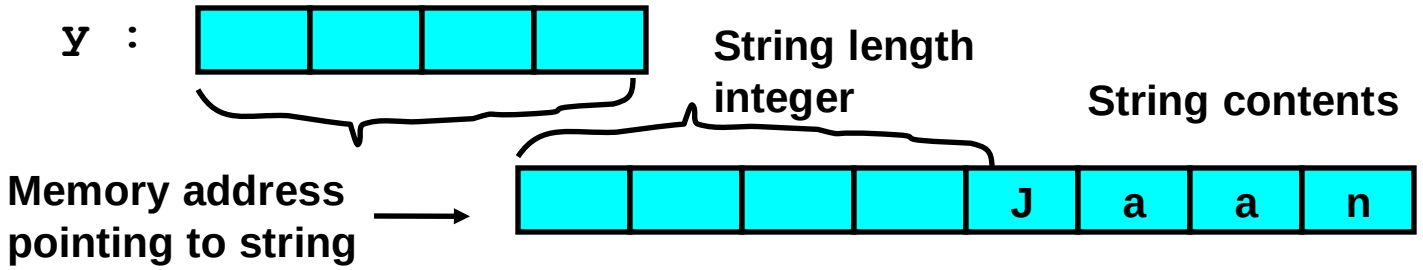
```
int x = 6;
```

then this "x" represents 4 bytes in memory holding value 6.



- If you want to create a string "Jaan Mets", it will also use up some memory and it will be placed at a some address (for example, 23423432 in memory)

```
String y = "Jaan";
```



Variables and objects in memory

- To summarise:
  - Variables of simple type hold the value directly
  - Variables of complex type hold a pointer to the value
  - The complex object is placed by the system somewhere in memory

- Therefore:
  - Declaring a simple type variable (int x) automatically creates place of value in memory
  - Declaring a complex type variable (String y) only creates a pointer
  - To actually create an object itself in memory one must say **`new`**

```
String y;
y = new String("Jaan Mets")
```

Important notes

- The arguments to `new` are different for different types
- You can always give a special value `null` to variable of a complex type
- While `new` is similar to `malloc` in C, you never need to say "`free`":
  - Useless objects in memory are thrown away automatically by Java
  - Object is useless, if no variables point to it

Example:

```
String y = new String("Jaan Mets");
y = null;
```

  - Automatic throwing away of objects in memory is called "Garbage collection" and is a part of most modern languages

Structures are represented by classes!

- The term “structure" is not used in Java. A structure is essentially the same as a Java object that has instance variables only (but no instance methods).
- Some other languages, which do not support objects in general, nevertheless do support structures.
- The data items in a structure -- in Java, its instance variables -- are called the fields of the structure. Each item is referred to using a field name.
- NB! The data items in the structure are
  - referred to by name
  - different fields in a structure are allowed to be of different types.
- For example, if the class `Person` is defined as:

```
class Person {
    String name;
    int age;
    float length;
 }
```

then an object of class `Person` could be considered to be a structure with **three fields.**

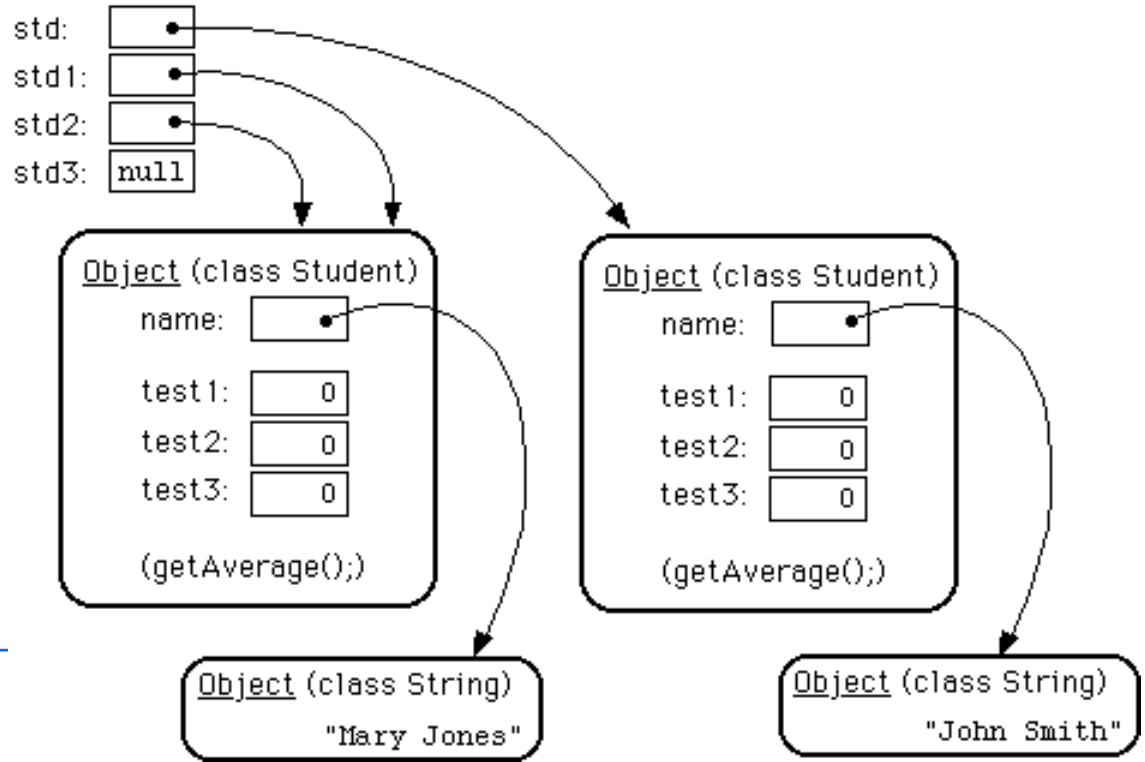
Using a class as a structure

```
public class mytest {
  class Person {
     String name;
     int age;
     float length;
  }

  public static void main(String[] args) {
    Person x = new Person();
    Person y = new Person();

    x.name = "Jaan Mets"
    x.age = 21;
    x.length = 1.87

    y.name = "Mihkel Unt"
    y.age = 35;
    y.length = 1.82
  ....
```

Another example

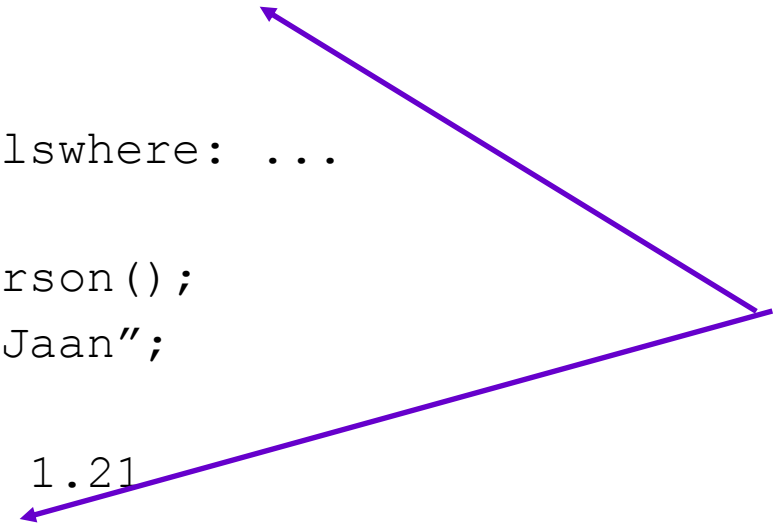
```
Student std, std1, std2, std3;
std = new Student();
std1 = new Student();
std2 = std1;
std3 = null;
std.name = "John Smith";
std1.name = "Mary Jones";
```

When one object variable is assigned to another, only a reference is copied. The object referred to is not copied.



We can put functions in classes as well!

```
class Person {
    String name;
    int age;
    float length;

    void show() {
      System.out.println(name+" "+age+" "+length);
    }
}

...  use elswhere: ...

x = new Person();
x.name = "Jaan";
x.age = 15
x.length = 1.21
x.show();
```



A function in a class (ie part of an object) is called a method

Objects: combine data with subroutines

- One common format for software modules is to contain some data, along with some subroutines for manipulating that data.
- For example, a mailing-list module might contain a list of names and addresses along with a subroutine for adding a new name, a subroutine for printing mailing labels, and so forth.
- Modules that could support this kind of information-hiding became common in programming languages in the early 1980s. Since then, a more advanced form of the same idea has more or less taken over software engineering.
- This approach is called object-oriented programming, often abbreviated as OOP.

Objects: state and message

- The central concept of object-oriented programming is the object, which is a kind of module containing data and subroutines.
- The point-of-view in OOP is that an object is a kind of self-sufficient entity that has an internal state (the data it contains) and that can respond to messages (calls to its subroutines).
- A mailing list object, for example, has a state consisting of a list of names and addresses.
  - If you send it a message telling it to add a name, it will respond by modifying its state to reflect the change.
  - If you send it a message telling it to print itself, it will respond by printing out its list of names and addresses.

Strings and arrays are somewhat special

- Strings and arrays are both just objects.
- However:
  - Since they are very common, there is a simplified way to create them.
  - There are many “built-in” methods and functions for both.
  - Array elements are accessed by a standard array syntax, like in C.
- NB! Differently from C:
  - A string in Java is not just a an array of characters.
  - The length of arrays and string is always kept along with contents

Strings again…

- Simple example:

```
String s = ”abc”;
System.out.println(s);
```

- Strings are built-in complex objects with their own methods
- Since strings are very common, there is a simplified format for creating them

```
String y = “oops”    // notice: no “new” construction!
int x = y.length(); // gives length of string “oops”
”abc”.length();     // gives the length of string “abc”
```

Array

- An array is just a sequence of items.
- The items in an array are numbered, and individual items are referred to by their position number.
- All the items in an array must be of the same type. So, the definition of an array is a numbered sequence of items, which are all of the same type.
- The number of items in an array is called the length of the array. The position number of an item in an array is called the index of that item. The type of the individual elements in an array is called the base type of the array.

- Java arrays are objects.
  - Arrays are created using the `new` operator.
  - No variable can ever hold an array; a variable can only refer to an array.
  - Any variable that can refer to an array can also hold the value `null`, meaning that it doesn't at the moment refer to anything.
  - An array belongs to a class, which like all classes is a subclass of the class `Object`.

Array

- Suppose that `A` is a variable that refers to an array.
  - The first item is `A[0]`,
  - the second is `A[1]`,
  - … and so forth.

- `A[k]` can be used just like a variable.

  You can assign values to it, you can use it in expressions, and you can pass it as a parameter to subroutines.

- Syntax:

  array-variable [integer-expression]

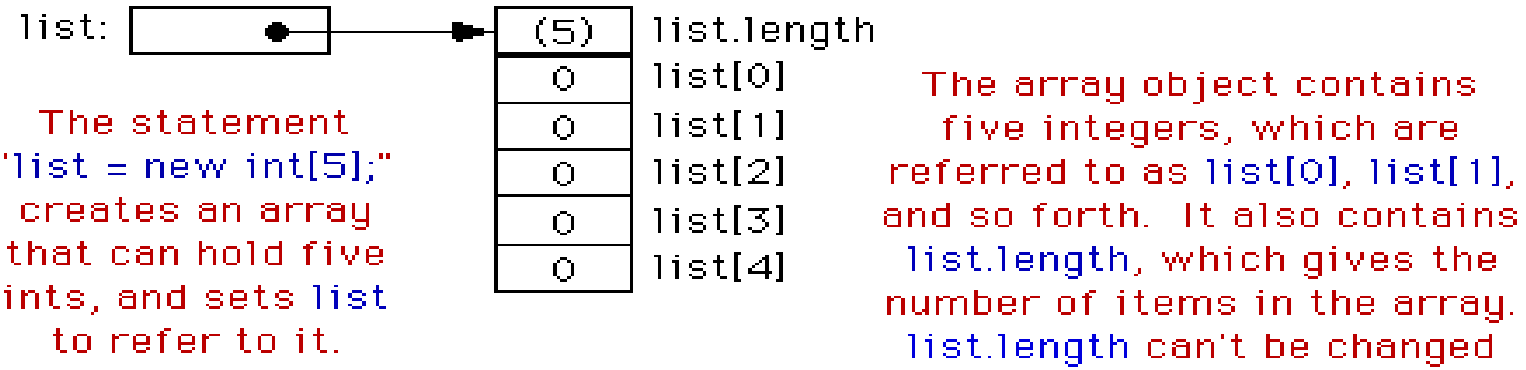
  for referring to an item in an array.

Array creation

- `int[] list;`

  creates a variable named `list` of type `int[]`.
- This variable is capable of referring to an array of `int`s, but initially its value is `null`
- The `new` operator is used to create a new array object, which can then be assigned to `list`.
- An example:

```
list = new int[5];
```



Array creation, elements

- Newly created array of integers is automatically filled with zeros.
- New array is always filled with a known, default value:
  - zero for numbers,
  - `false` for boolean,
  - the character with Unicode number zero for `char`,
  - `null` for objects.

- And the following loop would print all the integers in the array:

```
for (int i = 0; i < list.length; i++) {
     System.out.println( list[i] );
}
```

Array initialisation

- You can declare the variable and initialize it in a single step. For example,

```
int[] list = new int[5];
```

- Java also provides a way to initialize an array variable with a new array filled with a specified list of values:

```
int[] list = { 1, 4, 9, 16, 25, 36, 49 };
```

Processing an array

- Example:

```
double sum = 0;  // start with 0
for (int i = 0; i < A.length; i++)
   sum += A[i];
```

- Example:

```
int count = 0;
for (int i = 0; i < A.length; i++) {
   if (A[i] < 0.0)
      count++;
```

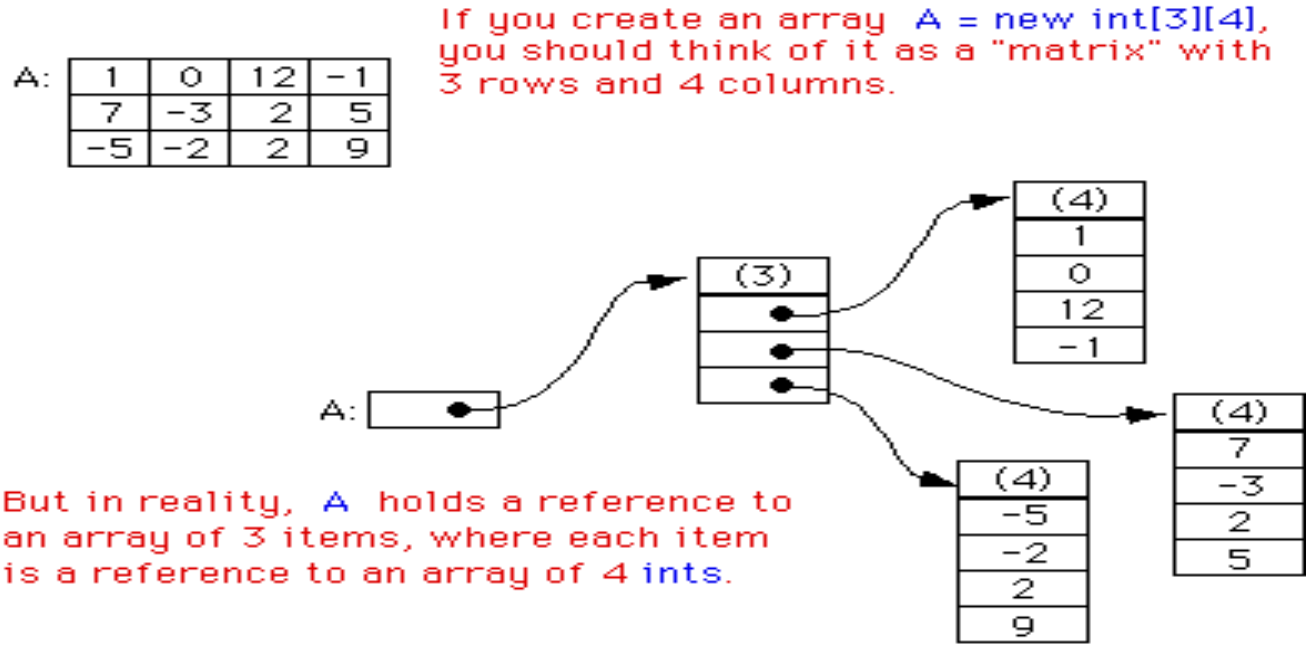
- Example:

```
 double max = A[0];
 for (int i = 1; i < A.length; i++) {
     if (A[i] > max)
        max = A[i];
 }
```

Two-dimensional arrays

- `int[][] A = new int[3][4]`

  declares a variable, `A`, of type `int[][]`, and it initializes that variable to refer to a newly created object.
- That object is an array of arrays of `ints`.
- The notation `int[3][4]` indicates that there are 3 arrays-of-`ints` in the array `A`, and that there are 4 `ints` in each of those arrays.



Two-dimensional array

```
A[0][0]    A[0][1]    A[0][2]    A[0][3]
A[1][0]    A[1][1]    A[1][2]    A[1][3]
A[2][0]    A[2][1]    A[2][2]    A[2][3]
```

```
int[][]  A  =  {  {  1,  0, 12, -1 },
                  {  7, -3,  2,  5 },
                  { -5, -2,  2,  9 }
               };
```